Python. Файлы. Строки. CodeStyle

Лекция 4

Преподаватель: *Дмитрий Косицин*

Модули и пакеты

Работа с командной строкой

Стиль кода

Строки

Файлы

Полезные ссылки и ответы

Модули и пакеты

Схема импорта модулей

При вызове «import x» происходит следующее:

1. find module
2. load module # an object loaded_module is created
   - create an object type of ModuleType
   - read source
   - compile source
   - execute source
3. sys.modules['x'] = loaded_module
4. <this_module>.x = loaded_module

Загрузка и перезагрузка модуля

reload(loaded_module) – перезагружает модуль, не создавая новый объект.

Сравните:

```
>>> from module import x
>>> # then use x directly

>>> import module
>>> # then use module.x
```

Важно! Это не то же самое, что удалить модуль и заново загрузить. В случае **reload** (importlib.reload в Python 3) все объекты в модуле *пересоздаются*! Более того, reload имеет множество подводных камней.

Загрузка модулей

При загрузке создаются и вызываются default-значения функций (вопрос: что произойдет?):

```
>>> def f(x=g()):
>>>     pass

>>> def g():
>>>     return 0
```

Для изменения поведения при исполнении модуля от поведения при импорте используется следующее:

```
>>> if __name__ == '__main__':
>>>     # code to be executed if module is an entry point
```

Импорт модулей

Модуль можно загружать по имени:

```
>>> import importlib
>>> module_instance = importlib.import_module('module_name')
```

Есть возможность загружать source, compiled (.**pyc**) и dynamic (.**pyd**, .**so**) модули по полному пути (см. ***imp*** в Python 2 и ***importlib.util*** в Python 3)

Очередность загрузки:

- package
- module
- namespace (PEP 420, Python 3.3+)

Пакеты и пространства имен

Package – папка с модулями, где присутствует файл *__init__.py*.

Namespace – файл *__init__.py* отсутствует.

Разница будет для вложенных папок: package вложенный обнаружится, а для namespace нужно явно прописать путь в **sys.path**.

Важно! *sys.path.append(x)* отличается от *sys.path += x*

Замечание. Узнать имя файла из модуля можно обратившись к переменной **__file__**, имя модуля – к **__name__**.

Дополнительные возможности

Модули можно загружать прямо из *zip*-архива.

Можно использовать относительные импорты (PEP-328).

Для **moduleX.py** верны следующие относительные импорты:

```
package/
   __init__.py
   subpackage1/
      __init__.py
      moduleX.py
      moduleY.py
   subpackage2/
      __init__.py
      moduleZ.py
   moduleA.py
```

```
from .moduleY import spam
from .moduleY import spam as ham
from . import moduleY
from ..subpackage1 import moduleY
from ..subpackage2.moduleZ import eggs
from ..moduleA import foo
from ...package import bar
from ...sys import path
```

Дополнительные возможности. Замечания

Относительные импорты не столь распространены ввиду худшей переносимости.

У модуля также может присутствовать ***docstring*** – его следует располагать вверху файла в тройных кавычках.

Модуль **__future__** является директивой компилятору создать **.pyc** файл, используя другие инструкции.

Существуют дополнительные механизмы: path hooks, metapath, module finders and loaders, etc.

Работа с командной строкой

Парсинг аргументов командной строки

```
import argparse

parser = argparse.ArgumentParser(description='Process some
integers.')

parser.add_argument('integers', type=int, nargs='+',
                    help='an integer for the accumulator')
parser.add_argument('--sum', dest='accumulate',
                    action='store_const',
                    const=sum, default=max,
                    help='sum the integers (default: find the
max)')

args = parser.parse_args()
print(args.accumulate(args.integers))
```

Для парсинга есть **ArgumentParser** в модуле **argparse**.

Методы **parse_args** и **parse_known_args** принимают некоторый список аргументов (по умолчанию **sys.argv**), парсят его и возвращают объект **Namespace**.

Произвольную строку запуска можно разбить на список с помощью модуля **shlex**.

Информации об аргументе добавляется с помощью **add_argument**:

- имена переменных через ‘-’ (dash), ‘--’ (double dash) или без них
- *dest* – имя переменной, в которой хранится значение
- *type* – преобразование типа
- *action* – действие при получении аргумента (*store*, *store_true*, *append*, etc.)
- *nargs* – количество аргументов (1, 1 и более, 0 и более)
- *default* – значение по умолчанию (если не передан)
- *required* – обязательный аргумент
- *choices* – список возможных значений
- *help* – описание аргумента

Стиль кода

...

Стиль кода

О пробелах

- Используйте ровно 4 пробела вместо tab.
- Добавляйте 2 строки между глобальными функциями, одну - между методами классов.
- В конце файла желательно добавлять пустую строку (так diff симпатичнее).
- Знаки математических операций следует выделять пробелами, кроме знака '=' при передаче аргументов по умолчанию
- Перед и после скобок (при вызове функции, индексировании) пробел не нужен
- После запятой пробел нужен всегда, перед – никогда

Стиль кода

Именование

var_name, function_name, ClassName, CONST_NAME

Переменные должны иметь понятные имена и доносить либо назначение переменной, либо описывать то, что в ней содержится (на что указывает).

В именах функций должен присутствовать глагол.

Длина строк

Длина строк не должна превышать 99 символов (почти всегда), документации – 72

Стиль кода

Дублирующийся код следует выносить в функции.

Комментарии нужны для пояснения тонких моментов.

Кодировка файлов желательна utf-8. Можно дополнительно добавить строку в начало файла:

```
# -*- coding: utf-8 -*-
```

Замечание: все требования – выдержка из PEP-8 и Google style guide.

Утилиты для проверки

Проверка кода без выполнения:

- pep8.py
- PyChecker
- PyFlakes
- pylint
- (PyCharm)

Строки

Строки

Python 2	Python 3
str – кортеж байтов **bytearray** – изменяемая последовательность байтов **unicode** – unicode строка	**bytes** – кортеж байтов **bytearray** изменяемая последовательность байтов **str** – unicode строка

Могут быть заданы как в одинарных '', так и двойных "" кавычках. Есть также строки в трех кавычках подряд (***docstring***).

К строкам могут быть применены литералы: **r**"", **u**"", **b**""

Методы работы со строками

Методы работы как с кортежами: сложение, умножение, проверка вхождения, длина, индексация:

```
>>> x = 'abc'
>>> 'b' in x
True

>>> x += 'd' * 2  # x: 'abcdd'
>>> len(x)
5

>>> x[:2]
ab
```

Вопрос: какой алгоритм используется для поиска вхождений?

Методы работы со строками

Поиск подстрок: **index**, **count** и **find/rfind**

```
>>> x.find('z')
-1
```

Проверка вхождения: **startswith, endswith**

```
>>> x.endswith(('aa', 'dd'))  # можно только один аргумент
True
```

Замена подстрок

```
>>> x = x.replace('a', 'z')  # 'zbcdd'; return a new one
```

Объединение/разбиение по символу: **join**, **split/rsplit** (см. max_split аргумент)

```
>>> 'c'.join(x.split('c'))  # 'zbcdd' -> ['zb', 'dd'] -> 'zbcdd'
zbcdd
```

Методы работы со строками

Важно! Если требуется объединить большое количество строк, следует использовать ''.**join**(список_строк)

Очистка строк: **strip/lstrip/rstrip**

```
>>> x.strip('d')
zbc
```

Преобразование регистра: **lower**, **upper**, **title**, **capitalize**.
Кодирование: **encode**, **decode**.
Проверка символов: **isalpha**, **isdigit**, etc.

Замечание. Строки, содержащие все цифры или все знаки пунктуации определены в модуле string.

Замечание. Работа с **base64** организуется с помощью библиотеки base64.

Форматирование строк

Подерживается *printf*-style форматирование:

```
>>> 'number of %.3f values in %s is %d' % (0.1234, 'some object', 3)
number of 0.123 values in some object is 3
```

Есть возможность использовать именованные аргументы:

```
>>> 'number of %(name)s is %(count)d' % {'name': 'names', 'count': 2}
number of names is 2
```

Поддерживается новый стиль форматирования строк:

```
>>> 'number of {0:.3f} values in {1} is {2}'.format(
        0.1234, 'some object', 3)
number of 0.123 values in some object is 3
```

Форматирование строк

А еще можно:

- аналогично использовать именованные аргументы: `"{name}"`
- индексировать аргументы: `"{items[0]}"`
- обращаться к атрибутам: `"{point.x}"`
- опускать индексы: `"{}{}"`
- повторять и менять местами индексы: `"{1}{0}{1}"`

Вопрос: что будет, если не совпадает количество аргументов для подстановки? А если нету такого именованного аргумента?

Форматирование строк

Замечание. Если вам нужно подставить множество локальных переменных, можно использовать словари **locals**() и **globals**(), определенные интерпретатором:

```
>>> x = 2
>>> "{x}".format(**locals())
2
```

Вопрос: верно ли, что так хитро можно менять значения локальных переменных?

Замечание. В Python 3 добавлен метод **format_map**, чтобы передавать словарь не распаковывая.

Файлы

Файлы

```
>>> f = open("path_to_file")  # file object
>>> # some actions
>>> f.close()
```

Если произойдет исключение до метода **close**, файл не будет закрыт.

Используйте менеджер контекстов (context manager):

```
>>> with open("path_to_file") as f:
>>>     # some actions
```

Полезные ссылки и ответы

Ответы

Строки модуля интерпретируются последовательно. При попытке создания объекта **f** – функции аргументы по умолчанию будут также интерпретированы и сохранены в данном объекте. Поскольку функция **g** объявлена ниже, произойдет исключение **NameError**, что такого имени нету. Обратите внимание, что значения аргументов по умолчанию создаются ***только один раз*** при загрузке модуля.

В стандартной библиотеке для поиска подстрок в строке используется алгоритм Бойера-Мура.

Если количество аргументов для подстановки не совпадает с количеством в шаблоне или один из требуемых именованных аргументов не передан, произойдет исключение **TypeError**.

Однако для подстановки можно передать словарь, в котором значений больше, чем требуется. Ошибки в таком случае не будет.

Полезные ссылки

Ответы

Модифицировать локальные переменные через словарь **locals**() нельзя, только получать значения.

Полезные ссылки

Подробнее механизм импортов описан здесь (Python 3):

- https://docs.python.org/3.7/library/modules.html
- https://docs.python.org/3.7/tutorial/modules.html
- https://docs.python.org/3.7/reference/import.html

Для Python 2 документация расположена по следующим ссылкам:

- https://docs.python.org/2.7/library/modules.html
- https://docs.python.org/2.7/tutorial/modules.html